MULTIMEDIA UNIVERSITY

STUDENT ID NO

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

# MULTIMEDIA UNIVERSITY

# FINAL EXAMINATION

**TRIMESTER 2, 2018/2019**

## ECE3166 ADVANCED MICROPROCESSORS

(All Majors)

05 MARCH 2019
9.00 AM – 11.00AM
(2 Hours)

---

**INSTRUCTIONS TO STUDENTS**

1. This Question paper consists of TEN pages with FOUR Questions only.
2. Attempt **ALL** questions. All questions carry equal marks and the distribution of the marks for each question is given.
3. Please print all your answers in the Answer Booklet provided.

**Question 1**

(a) Moore's law has been widely accepted as a goal for the semiconductor industry.

(i) State Moore's law. [1 mark]

(ii) Describe one of the key challenges of following Moore's law for future nanoscale transistors. [1 mark]

(b) State the application areas of the Arm Cortex-A and Cortex-M series to show their differences. [2 marks]

(c) Provide TWO differences between protected and real mode from a programmer's view point. [2 marks]

(d) Table Q1 shows the memory contents in hexadecimal in the 80386DX (Little Endian) microprocessor.

Table Q1

| Address | Memory content (hex) |
|---|---|
| 4000H | 1E |
| 4001H | AB |
| 4002H | A1 |
| 4003H | 5C |

(i) State the double word stored in the memory starting at address 4000H in hexadecimal. [2 marks]

(ii) From the hexadecimal integer obtained in (i), state the equivalent decimal integer. [1 mark]

(e) Redraw the diagram in Figure Q1 on your answer sheet and complete it to show the software architecture of an 8086 microprocessor. The diagram must contain the following:

(i) Various internal registers. For the general purpose registers, include their 8-bit registers for byte data operations. [9 marks]

(ii) Memory and I/O peripherals blocks. State their beginning and ending addresses. [3 marks]

(iii) Show the pointers from the relevant internal registers to the various memory segments. Name and state the size of the memory segments. [4 marks]

**Continued...**

**Question 1 (Continued...)**

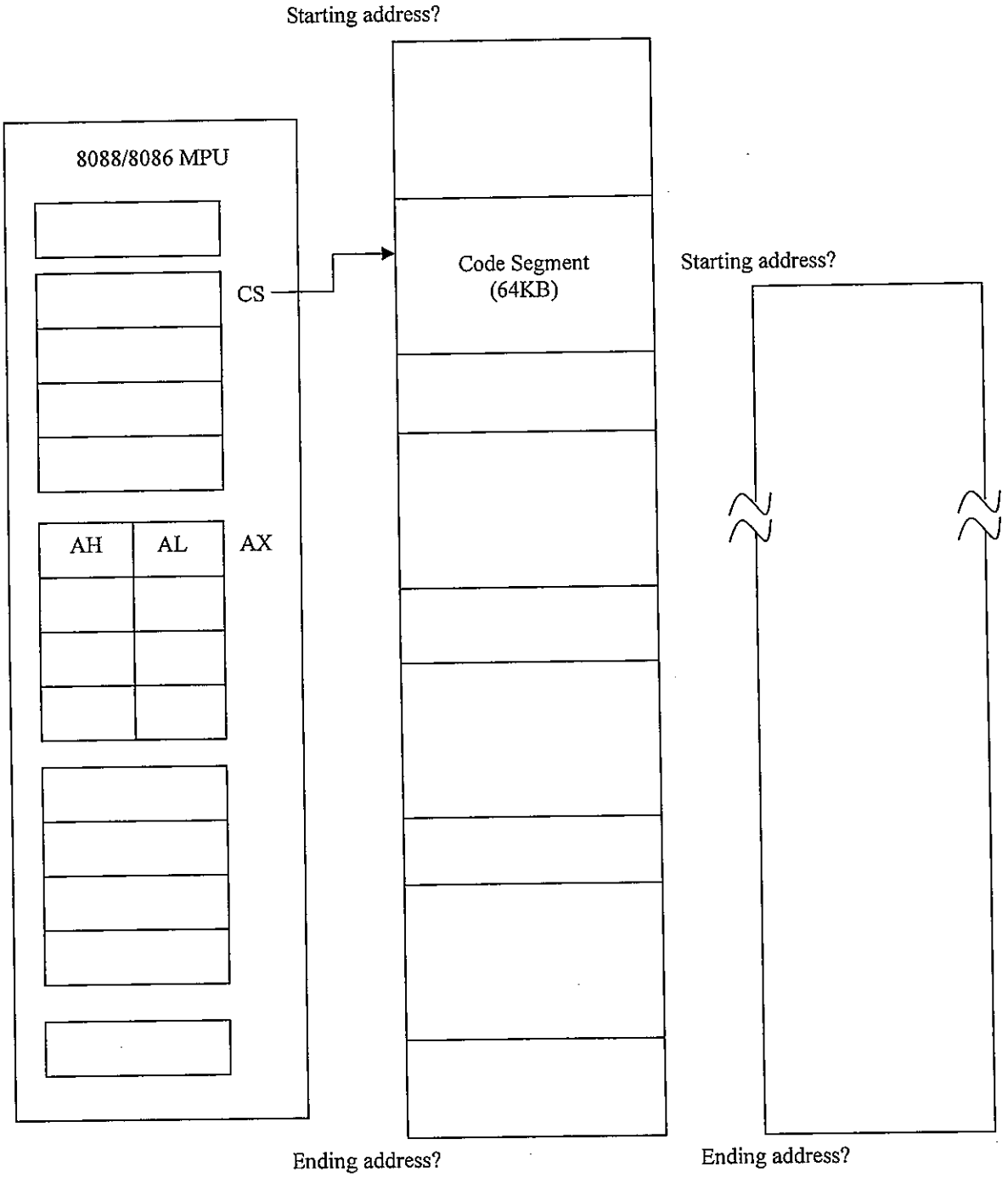



Figure Q1

**Question 2**

(a) Find the unknown value for the logical addresses corresponding to each of the following physical addresses. All numbers are hexadecimal numbers.

(i) A200H : 0234H = ? [2 marks]

(ii) ? : 9846H = 23056H [2 marks]

**Continued...**

**Question 2 (Continued...)**

(b) The contents of various registers in an 80386DX microprocessor are given in Table Q2. Identify the **addressing mode** and compute the **physical address** (real-mode programming) of the destination operand for each of the following instructions. Show your calculations.

(i) MOV [DI], 0ABCDH [3 marks]

(ii) MOV GS:[BX] + 20H, AH [3 marks]

(iii) MOV [BX][DI] + 5H, CX [3 marks]

Table Q2

| Register | Contents |
|---|---|
| AX | 4444H |
| BX | 0050H |
| CX | ABCDH |
| DX | 2010H |
| SI | 0020H |
| DI | 0030H |

| Register | Contents |
|---|---|
| CS | 0600H |
| DS | 0700H |
| ES | 0800H |
| FS | 0900H |
| GS | 1000H |
| SS | 1100H |

(c) Determine the contents of the affected memory locations after each of the following instructions has been executed. Assume that the contents of the registers are as shown in Table Q2.

(i) AND AX, CX [2 marks]

(ii) XOR DH, 67H [1 marks]

(d) Analyze the following instruction sequence:

```
MOV AX, 0203H
MOV BX, 0403H
MOV CX, 0605H
MOV DX, F00FH
ADD BL, AL
SUB CX, AX
MUL BH
NOT DX
```

What will be the new contents of registers AX, BX, CX and DX after this instruction sequence is executed? [6 marks]

(e) Complete the partial x86 assembly program (real-mode programming) shown in Figure Q2 to store the value 0FH into each element (byte wide) of a string starting at address ES: 3020H to ES:3000H. [3 marks]

```
        ...
    MOV DI, 3020H
        ...
    REP STOSB

; Note that "..." may represent one or more assembly instructions.
```

Figure Q2

**Continued...**

**Question 3**

(a) The content of the Global Descriptor Table Register (GDTR) is 3300 0800 2FFFH.

(i) Determine the starting address, ending address and the size of the table. [3 marks]

(ii) Calculate the number of descriptors that can be stored in this table. [2 marks]

(iii) If the value of 1120H is loaded to the Task Register (TR), determine the address range of the Task State Segment (TSS) descriptor. [3 marks]

(b) A code segment selector register is loaded with a value of 2227H. The base addresses of Global Descriptor Table (GDT) and Local Descriptor Table (LDT) are 0400 1000H and 0600 3600H, respectively. The contents of the memory are shown in Table Q3.

Table Q3

| Linear address | Contents | | Linear address | Contents |
|---|---|---|---|---|
| 0400 3220H | 00FFH | | 0600 5820H | 22FFH |
| 0400 3222H | 0020H | | 0600 5822H | C500H |
| 0400 3224H | 9210H | | 0600 5824H | FBA0H |
| 0400 3226H | 0500H | | 0600 5826H | 0800H |

(i) Find the Requested Privilege Level (RPL) and identify whether the descriptor is stored in the Global Descriptor Table (GDT) or Local Descriptor Table (LDT). [2 marks]

(ii) Compute the address range of the code segment descriptor. [3 marks]

(iii) Determine the base address and the size of the segment. [4 marks]

(c) Assume that 32-bit paging is enabled on an IA-32 processor with a page size of 4KB. The linear address of an operand is 04301820H, while the contents of the corresponding page directory entry and page table entry are 06002007H and 11002005H, respectively.

(i) Compute the physical address range of the page table. [2 marks]

(ii) Given that the base address of the page directory is 00002000H, determine the physical address range of the page directory entry and the page table entry. [6 marks]

**Continued...**

**Question 4**

(a) Data dependancies may cause stalling during the pipelining of a microprocessor. Sketch a pipelined execution of these three instructions. Then, highlight the number of cycles the second ADD instruction need to be stalled, assuming each instruction requires one cycle per stage to complete each set of fives-stage instruction execution.

```
ADD         R0, R1 #100
SUBTRACT    R2, R3, #50
ADD         R4, R2, #25
```

[6 marks]

(b) Listing Q4 shows four SSE instructions. Show your final answers in hexadecimal.

```
struct SVector
{  float a, b, c, d;};
void SSE_Instr(struct SVector op0, struct SVector op1 )
{
   __asm
      {      MOVUPS XMM0, op0
            //XMM0 = 40900000 C0600000 41B20000 41340000 H
            //       (4.5)    (-3.5)   (22.25)   (11.25)

            MOVUPS XMM1, op1
            //XMM1 = 41000000 40A80000 C1200000 41180000 H
            //       (8)      (5.25)   (-10)    (9.5)

            MAXPS  XMM0, XMM1

            ADDSS  XMM1, XMM0
      }
}
int main()
{   struct SVector obj0, obj1;
    obj0.a = 11.25; obj0.b = 22.25; obj0.c = -3.5; obj0.d = 4.5;
    obj1.a = 9.5;   obj1.b = -10;   obj1.c = 5.25; obj1.d = 8;
    SSE_Instr(obj0, obj1);
    return 0;
}
```

Listing Q4

(i) Analyze the program and state the 128-bit content of the destination operand after execution of the MAXPS instruction.

[4 marks]

(ii) Evaluate the changes of the the destination operand after execution of the ADDSS instruction. Show the floating point to single-precision real number conversion involved in detail. Then, state the final content of the destination operand.

[9 marks]

(c) With the aid of THREE diagrams, contrast threading in early processors, processors with hyper-threading technology and multicore processors.

[6 marks]

**End of paper.**

## Appendix A

### Instruction Set Summary for Intel 8086/80186/80286/80386/80486

AAA - Ascii Adjust for Addition
AAS - Ascii Adjust for Subtraction
ADC - Add With Carry
ADD - Arithmetic Addition
AND - Logical And
BSF - Bit Scan Forward (386+)
BSR - Bit Scan Reverse (386+)
BSWAP - Byte Swap (486+)
BT - Bit Test (386+)
BTC - Bit Test with Compliment (386+)
BTR - Bit Test with Reset (386+)
BTS - Bit Test and Set (386+)
CALL - Procedure Call
CBW - Convert Byte to Word
CDQ - Convert Double to Quad (386+)
CLC - Clear Carry
CLD - Clear Direction Flag
CLI - Clear Interrupt Flag (disable)
CLTS - Clear Task Switched Flag (286+ privileged)
CMC - Complement Carry Flag
CMP - Compare
CMPS - Compare String (Byte, Word or Doubleword)
CMPXCHG - Compare and Exchange
CWD - Convert Word to Doubleword
CWDE - Convert Word to Extended Doubleword (386+)
DAA - Decimal Adjust for Addition
DAS - Decimal Adjust for Subtraction
DEC - Decrement
DIV - Divide
ENTER - Make Stack Frame (80188+)
ESC - Escape
HLT - Halt CPU
IDIV - Signed Integer Division
IMUL - Signed Multiply
IN - Input Byte or Word From Port
INC - Increment
INT - Interrupt
INTO - Interrupt on Overflow
INVD - Invalidate Cache (486+)
INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+)
IRET/IRETD - Interrupt Return
Jxx - Jump Instructions Table
JCXZ/JECXZ - Jump if Register (E)CX is Zero
JMP - Unconditional Jump
LAHF - Load Register AH From Flags
LAR - Load Access Rights (286+ protected)
LDS - Load Pointer Using DS
LEA - Load Effective Address
LES - Load Pointer Using ES
LFS - Load Pointer Using FS (386+)
LGDT - Load Global Descriptor Table (286+ privileged)
LIDT - Load Interrupt Descriptor Table (286+ privileged)
LGS - Load Pointer Using GS (386+)
LLDT - Load Local Descriptor Table (286+ privileged)
LMSW - Load Machine Status Word (286+ privileged)
LOCK - Lock Bus
LODS - Load String (Byte, Word or Double)

LOOP - Decrement CX and Loop if CX Not Zero
LOOPE/LOOPZ - Loop While Equal / Loop While Zero
LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal
LSL - Load Segment Limit (286+ protected)
LSS - Load Pointer Using SS (386+)
LTR - Load Task Register (286+ privileged)
MOV - Move Byte or Word
MOVS - Move String (Byte or Word)
MOVSX - Move with Sign Extend (386+)
MOVZX - Move with Zero Extend (386+)
MUL - Unsigned Multiply
NEG - Two's Complement Negation
NOP - No Operation (90h)
NOT - Logical inversion
OR - Inclusive Logical OR
OUT - Output Data to Port
POP - Pop Word off Stack
POPA/POPAD - Pop All Registers onto Stack (80188+)
POPF/POPFD - Pop Flags off Stack
PUSH - Push Word onto Stack
PUSHA/PUSHAD - Push All Registers onto Stack (80188+)
PUSHF/PUSHFD - Push Flags onto Stack
RCL - Rotate Through Carry Left
RCR - Rotate Through Carry Right
REP - Repeat String Operation
REPE/REPZ - Repeat Equal / Repeat Zero
REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero
RET/RETF - Return From Procedure
ROL - Rotate Left
ROR - Rotate Right
SAHF - Store AH Register into FLAGS
SAL/SHL - Shift Arithmetic Left / Shift Logical Left
SAR - Shift Arithmetic Right
SBB - Subtract with Borrow/Carry
SCAS - Scan String (Byte, Word or Doubleword)
SETS - Set if Signed (386+)
SETNS - Set if Not Signed (386+)
SETC - Set if Carry (386+)
SETNC - Set if Not Carry (386+)
SETO - Set if Overflow (386+)
SETP/SETPE - Set if Parity / Set if Parity Even (386+)
SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+)
SGDT - Store Global Descriptor Table (286+ privileged)
SIDT - Store Interrupt Descriptor Table (286+ privileged)
SHL - Shift Logical Left
SHR - Shift Logical Right
SHLD/SHRD - Double Precision Shift (386+)
SLDT - Store Local Descriptor Table (286+ privileged)
SMSW - Store Machine Status Word (286+ privileged)
STC - Set Carry
STD - Set Direction Flag
STI - Set Interrupt Flag (Enable Interrupts)
STOS - Store String (Byte, Word or Doubleword)
STR - Store Task Register (286+ privileged)
SUB - Subtract
TEST - Test for Bit Pattern
WBINVD - Write-Back and Invalidate Cache (486+)
XCHG - Exchange
XOR - Exclusive OR

| Mnemonic | Meaning | Format | Operation | Flags Affected |
|---|---|---|---|---|
| MOVS | Move string | MOVSB/MOVSW | ((ES)0 + (DI)) ← ((DS)0 + (SI))<br>(SI) ← (SI) ± 1 or 2<br>(DI) ← (DI) ± 1 or 2 | None |
| CMPS | Compare string | CMPSB/CMPSW | Set flags as per<br>((DS)0 + (SI)) − ((ES)0 + (DI))<br>(SI) ← (SI) ± 1 or 2<br>(DI) ← (DI) ± 1 or 2 | CF, PF, AF, ZF, SF, OF |
| SCAS | Scan string | SCASB/SCASW | Set flags as per<br>(AL or AX) − ((ES)0 + (DI))<br>(DI) ← (DI) ± 1 or 2 | CF, PF, AF, ZF, SF, OF |
| LODS | Load string | LODSB/LODSW | (AL or AX) ← ((DS)0 + (SI))<br>(SI) ← (SI) ± 1 or 2 | None |
| STOS | Store string | STOSB/STOSW | ((ES)0 + (DI)) ← (AL or AX)<br>(DI) ← (DI) ± 1 or 2 | None |

| Prefix | Used with: | Meaning |
|---|---|---|
| REP | MOVS<br>STOS | Repeat while not end of string<br>CX ≠ 0 |
| REPE/REPZ | CMPS<br>SCAS | Repeat while not end of string and strings are equal<br>CX ≠ 0 and ZF = 1 |
| REPNE/REPNZ | CMPS<br>SCAS | Repeat while not end of string and strings are not equal<br>CX ≠ 0 and ZF = 0 |

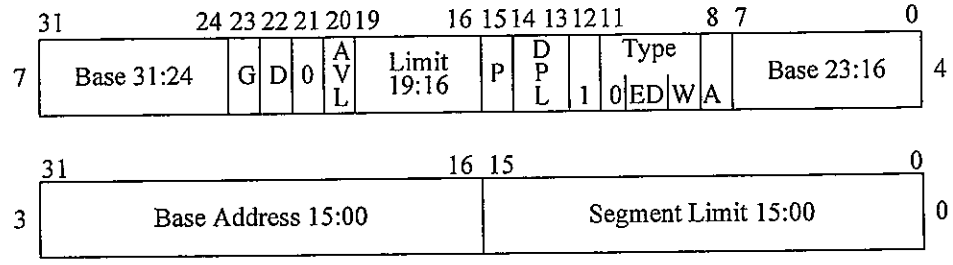
## Appendix B

### Summary of Selected SSE Instructions

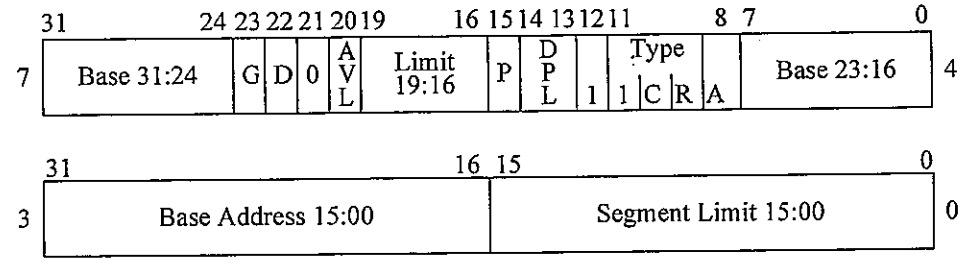
| Mnemonic | Description |
|---|---|
| MOVAPS | Move aligned packed single-precision floating-point values |
| MOVUPS | Move unaligned packed single-precision floating-point values |
| MOVHPS | Move high packed single-precision floating-point values |
| MOVHLPS | Move packed single-precision floating-point high to low |
| MOVLHPS | Move packed single-precision floating-point low to high |
| MOVLPS | Move low packed single-precision floating-point values |
| MOVMSKPS | Move packed single-precision floating-point mask |
| MOVSS | Move scalar single-precision floating-point values |
| ADDSS | Scalar add pair of operands |
| ADDPS | Packed add pairs of operands |
| SUBSS | Scalar subtract pair of operands |
| SUBPS | Packed subtract pair of operands |
| MINPS | Packed return the minimum of the pair of operands |
| MAXPS | Packed return the maximum of the pair of operands |

## Appendix C

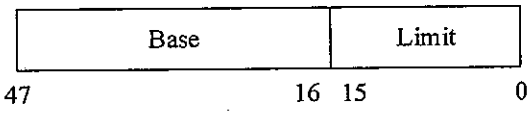
### Data-Segment Descriptor:



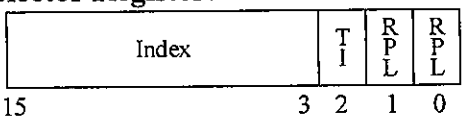

### Code-Segment Descriptor:



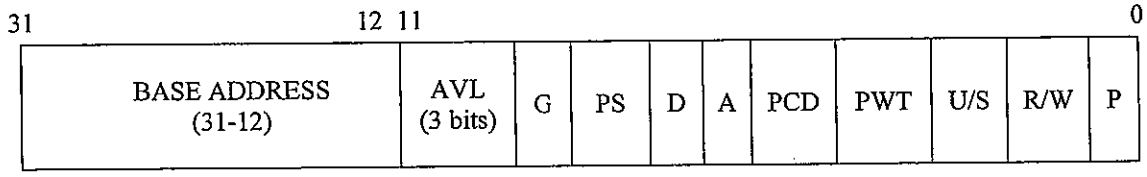

### Global Descriptor Table Register (GDTR) and Interrupt Descriptor Table Register (IDTR):




### Selector Register:




### Page Directory or Page Table Entry:




### Linear Address: